

Header नाम का एक functional component बनाया गया है, जो props के रूप में searchTerm, setSearchTerm और onSearch लेता है।

```
js
CopyEdit
const [localSearch, setLocalSearch] = useState(searchTerm);
```

◆ localSearch नाम की state बनाई गई है, जो input field का स्थानीय (local) value रखेगी।

```
js
CopyEdit
const [suggestions, setSuggestions] = useState([]);
```

◆ सर्च के सुझावों (suggestions) को रखने के लिए एक state बनाई गई है।

```
js
CopyEdit
const [selectedCategory, setSelectedCategory] = useState('All');
```

◆ चुनी हुई category को track करने के लिए state बनाई गई है, जिसकी default value 'All' है।

```
js
CopyEdit
const navigate = useNavigate();
```

◆ navigate फ़ंक्शन बनाया गया है ताकि programmatically routes बदले जा सकें।

useEffect – सर्च सुझावों को अपडेट करना:

```
js
CopyEdit
useEffect(() => {
  if (localSearch.trim() === '') {
    setSuggestions([]);
    return;
  }

  const filtered = allProducts.filter(item =>
    item.title?.toLowerCase().includes(localSearch.toLowerCase()) &&
    (selectedCategory === 'All' || item.type?.toLowerCase() ===
selectedCategory.toLowerCase())
```

```
);  
  
setSuggestions(filtered.slice(0, 5));  
}, [localSearch, selectedCategory]);
```

- ◆ जब भी localSearch या selectedCategory बदले, ये effect चलेगा।
 - ◆ अगर सर्च खाली है तो सुझाव खाली कर दिए जाएंगे।
 - ◆ नहीं तो products को filter किया जाएगा जो सर्च से मिलते हों और category से भी।
 - ◆ सुझावों को 5 आइटम तक सीमित किया गया है।
-

सर्च input के बदलने पर:

```
js  
CopyEdit  
const handleInputChange = (e) => {  
  setLocalSearch(e.target.value);  
  setSearchTerm(e.target.value);  
  onSearch(e);  
};
```

- ◆ जैसे ही user कुछ टाइप करता है, local और global दोनों search states अपडेट होती हैं।
 - ◆ साथ ही onSearch callback को call किया जाता है।
-

Suggestion क्लिक होने पर:

```
js  
CopyEdit  
const handleSuggestionClick = (title) => {  
  setLocalSearch(title);  
  setSuggestions([]);  
  
  navigate(`/search-results?query=${title}&category=${selectedCategory}`);  
};
```

- ◆ जब user किसी सुझाव पर क्लिक करता है तो सर्च value अपडेट होती है और उस title के साथ search results पेज पर भेजा जाता है।
-

🔍 सर्च बटन क्लिक पर:

js

CopyEdit

```
const handleSearchClick = () => {  
  if (localSearch.trim() !== '') {  
  
    navigate(`/search-results?query=${localSearch}&category=${selectedCategory}`);  
  }  
};
```

◆ अगर सर्च इनपुट खाली नहीं है तो, उसी query और category के साथ search results पेज पर नेविगेट किया जाता है।

🔧 JSX (UI Layout):

यहाँ से UI structure शुरू होता है:

js

CopyEdit

```
<div className="navbar">
```

◆ पूरी navbar को एक container div में रखा गया है।

◆ लोगो:

js

CopyEdit

```
<div className="nav-logo border">  
  <div className="logo"></div>  
</div>
```

◆ वेबसाइट का लोगो दिखाने के लिए placeholder div है।

📍 डिलीवरी लोकेशन:

js

CopyEdit

```
<div className="nav-address border">  
  <p className="add-first">Deliver to</p>  
  <div className="add-icon">  
    <FaLocationDot />  
    <p className="add-second">India</p>
```

```
    </div>
</div>
```

- ◆ यूज़र को "Deliver to India" लिखा हुआ दिखाया जाता है, साथ में location icon के साथ।

सर्च सेक्शन:

```
js
CopyEdit
<div className="nav-search">
```

- ◆ सर्च के लिए category dropdown, input box और आइकन हैं।

```
js
CopyEdit
<select>...</select>
```

- ◆ यूज़र category चुन सकता है – जैसे All, Mobile, Laptop वगैरह।

```
js
CopyEdit
<input />
```

- ◆ सर्च बॉक्स जिसमें user input देता है।

```
js
CopyEdit
<div className="search-icon" onClick={handleSearchClick}>
  <FaMagnifyingGlass />
</div>
```

- ◆ सर्च आइकन, जिस पर क्लिक करके सर्च किया जा सकता है।

```
js
CopyEdit
{ suggestions.length > 0 && (
  <ul className="suggestion-box">...</ul>
)}
```

- ◆ अगर suggestions हैं, तो उन्हें list में दिखाया जाता है।

Sign-in:

```
js
```

CopyEdit

```
<div className="nav-signin">
  <Link to="/register">...</Link>
</div>
```

◆ "Hello, sign in" और "Account & Lists" लिंक दिखाया गया है, जो रजिस्ट्रेशन पेज की ओर ले जाता है।

Return और Orders:

js

CopyEdit

```
<div className="nav-return">
  <p><span>Returns</span></p>
  <p className="nav-second">& Orders</p>
</div>
```

◆ Returns और Orders सेक्शन।

Cart:

js

CopyEdit

```
<div className="nav-cart border">
  <Link to="/cart">
    <FaCartShopping />
    <span>Cart</span>
  </Link>
</div>
```

◆ कार्ट आइकन और "Cart" टेक्स्ट, जो cart पेज की ओर ले जाता है।

js


CopyEdit

```
export default Header;
```

◆ Header component को बाहर एक्सपोर्ट किया गया है ताकि इसे अन्य components में इस्तेमाल किया जा सके।

App.js

└─ Header.js (props: searchTerm, setSearchTerm, onSearch)

 Step-by-Step Search Logic Explanation:

✓ 1. App.js में searchTerm की State

js

Copy

Edit

```
const [searchTerm, setSearchTerm] = useState("");
```

यह App component में searchTerm की central state है, जिससे पूरी ऐप में search से जुड़ी जानकारी maintain होती है।

🧠 2. App.js → Header को Props भेजता है

js

Copy

Edit

```
<Header  
  searchTerm={searchTerm}  
  setSearchTerm={setSearchTerm}  
  onSearch={handleSearch}  
/>
```

App.js में जो searchTerm और setSearchTerm हैं, उन्हें Header को prop के रूप में भेजा गया है।

साथ ही, एक callback function handleSearch भी भेजा गया है, जो search trigger करता है।

🔥 3. Header.js में searchTerm और setSearchTerm का इस्तेमाल

js

Copy

Edit

```
const [localSearch, setLocalSearch] = useState(searchTerm);
```

Header में एक local state localSearch बनाई गई है, जो input field की value को locally track करती है।

📄 4. User सर्च बॉक्स में कुछ टाइप करता है

js

Copy

Edit

```
const handleInputChange = (e) => {  
  setLocalSearch(e.target.value);    // local state update  
  setSearchTerm(e.target.value);    // parent state update (App.js)  
  onSearch(e);                      // parent के function को call करना  
};
```

इस function में तीन काम होते हैं:

इनपुट की value को localSearch में रखा जाता है।

वही value parent की searchTerm में भेज दी जाती है (setSearchTerm)।

फिर parent के search handler को call किया जाता है (onSearch(e)), जो App.js में defined है।

🔍 5. App.js का handleSearch function

js

Copy

Edit

```
const handleSearch = (e) => {  
  e.preventDefault();  
  console.log("Searching for:", searchTerm);  
};
```

यह बस एक डेमो है (फ़िलहाल सिर्फ console.log) लेकिन यहीं से actual search request (API call या navigate) की जा सकती है।

📄 6. Search Results Page को Props में searchTerm मिलता है

js

Copy

Edit

```
<Route path="/search-results" element={<SearchResultsPage searchTerm={searchTerm} />} />
```

जब यूज़र सर्च करता है और /search-results पेज पर जाता है, तो वहां भी searchTerm भेजा जाता है, जिससे result filter या fetch किए जा सकें।

🔄 Summary (Flowchart Style):

text

Copy

Edit

User types in Header → handleInputChange()

→ localSearch set होता है

→ App.js की setSearchTerm() call होती है

→ App.js का handleSearch() trigger होता है

→ Header से navigate होता है SearchResultsPage की ओर

→ SearchResultsPage को searchTerm prop के रूप में मिलता है

const handleSearch = (e) => {

- ये एक **function definition** है, जो किसी **search button** या **form submit event** पर call किया जाएगा।
- **e** का मतलब है **event object** — यानि वो object जिसमें click या submit से जुड़ी सारी जानकारी होती है।

🧠 जैसे ही user ने कुछ action किया (submit या search button click), ये function चलेगा।

✅ **e.preventDefault();**

- ये लाइन **browser का default behavior** रोकती है।

- Normally, अगर ये function किसी `<form onSubmit>` से जुड़ा है, तो form submit करते ही browser **page reload कर देगा**।
- हम reload नहीं चाहते क्योंकि हम खुद handle कर रहे हैं search को — इसलिए हम इसे रोक देते हैं।

🛑 Default submit को रोक दो — JS से control करेंगे!

✅ `console.log("Searching for:", searchTerm);`

- ये simply browser के **console** में **message print करता है**।
- इसमें हम देख सकते हैं कि user ने क्या search किया है।

🔍 Output होगा कुछ ऐसा:

yaml

CopyEdit

Searching for: iPhone

➡ यहाँ `searchTerm` एक variable है जिसमें search input की value होती है।

📦 पूरा का पूरा क्या कर रहा है ये function?

जब user search करता है:

- Browser को reload करने से रोका जाता है (using `e.preventDefault()`)
- और उस वक्त क्या search किया गया, वो console में दिखाया जाता है



📦 Full Code:

js

CopyEdit


```
useEffect(() => {
  if (localSearch.trim() === '') {
    setSuggestions([]);
    return;
  }

  const filtered = allProducts.filter(item =>
    item.title?.toLowerCase().includes(localSearch.toLowerCase()) &&
    (selectedCategory === 'All' || item.type?.toLowerCase() ===
selectedCategory.toLowerCase())
  );

  setSuggestions(filtered.slice(0, 5)); // 📌 यही updates the parent
suggestions
}, [localSearch, selectedCategory]);
```

🔍 Line-by-Line Explanation (हिन्दी में):

✅ `useEffect(() => {`

👉 ये React का hook है — जो तब चलता है जब dependencies (last line में दिए हुए variables) change होते हैं।

यहाँ ये effect हर बार **localSearch** या **selectedCategory** बदलने पर run होगा।

✅ `if (localSearch.trim() === '') {`

👉 चेक कर रहा है कि **search box** खाली है या नहीं
(`.trim()` का मतलब है leading/trailing spaces हटाना)

✅ `setSuggestions([]);`

👉 अगर search खाली है, तो **suggestion list** खाली कर दो

✅ `return;`

👉 और बाकी का filter logic चलाने की कोई ज़रूरत नहीं, सीधा बाहर निकल जाओ।

✅ `const filtered = allProducts.filter(item => ...);`

👉 ये line `allProducts` array से items को filter करती है
मतलब सिर्फ वो items निकालो:

1. जिनका title `localSearch` से match करता हो
 2. और selected category से match करता हो (अगर 'All' नहीं है)
-



`item.title?.toLowerCase().includes(localSearch.toLowerCase())`

👉 हर item का title **lowercase** में **convert** करके search term से match कर रहा है।
`?.` का मतलब है अगर `title` undefined हुआ तो error न दो।

✅ `(selectedCategory === 'All' || item.type?.toLowerCase() === selectedCategory.toLowerCase())`

👉 Category match करने का logic:

- अगर user ने **"All"** select किया है, तो सब चलेगा
 - वरना check करेगा कि item का type user की selected category से match करता है या नहीं
-

✅ `setSuggestions(filtered.slice(0, 5));`

👉 Filtered result में से सिर्फ **top 5 suggestions** दिखाओ
`slice(0, 5)` मतलब 0 से 4 index तक की values लेना

✅ `}, [localSearch, selectedCategory]);`

👉 ये dependency array है —
ये पूरा `useEffect` block तब ही दुबारा चलेगा जब:

- `localSearch` (search box की value)
- या `selectedCategory` (dropdown से चुनी category) बदलती है।

🧠 Summary:

काम	क्या हो रहा है
Empty search	Suggestions clear
Valid search	Matching products filter
Max suggestions	सिर्फ 5 items दिखाना
Auto update	जब भी search या category बदले

`handleSuggestionClick` - जब user suggestion पर click करता है

js

CopyEdit

```
const handleSuggestionClick = (title) => {  
  setLocalSearch(title);  
  setSuggestions([]);
```

```
  navigate(`/search-results?query=${title}&category=${selectedCategory}`);  
};
```

🔍 Explanation:

लाइन

क्या कर रही है

<code>setLocalSearch(title);</code>	जिस suggestion पर user ने क्लिक किया, उसे input box में डाल दो
<code>setSuggestions([]);</code>	Suggestion list को खाली कर दो (ताकि वो हट जाए screen से)
<code>navigate(...)</code>	User को <code>/search-results</code> वाले page पर भेज दो, और साथ में URL में query और category भी भेज दो — जिससे search result उस हिसाब से दिखे

🧠 ये function यूजर के suggestion selection को **final search** में convert करता है।

✅ 2. `handleSearchClick` - जब user manually search icon दबाए

js

CopyEdit

```
const handleSearchClick = () => {
  if (localSearch.trim() !== '') {

    navigate(`/search-results?query=${localSearch}&category=${selectedCategory}`);
  }
};
```

🔍 Explanation:

लाइन	क्या कर रही है
<code>if (localSearch.trim() !== '')</code>	पहले check कर रहे हैं कि input खाली तो नहीं
<code>navigate(...)</code>	अगर search term valid है, तो user को result page पर भेज दो और उसकी query + category साथ में भेज दो URL में

🧠 ये वाला function तब काम करता है जब user खुद search button दबाता है (suggestion पर click किए बिना)



Real-World Analogy (Short Story Mode):

Imagine कर, तू Amazon पे कुछ type कर रहा है जैसे "iPhone".
नीचे suggestions आए "iPhone 13", "iPhone 14"...

- ◆ तू "iPhone 14" suggestion पर click करता है — तो `handleSuggestionClick()` trigger होता है।
- ◆ और अगर तू manually search icon दबा देता है — तो `handleSearchClick()` चलता है।

Step 3: Suggestions को दिखाना

js

CopyEdit

```
{suggestions.length > 0 && (  
  <ul className="suggestion-box">  
    {suggestions.map((item) => (  
      <li  
        key={item.id}  
        onClick={() => handleSuggestionClick(item.title)}  
      >  
        {item.title}  
      </li>  
    ))}  
  </ul>  
)}
```

🧠 अगर suggestions हैं, तो उन्हें एक list की तरह दिखाया जाता है।
हर list item पर click करने पर user को उस product की detail/search-result पेज पर ले जाया जाता है।

🌟 Summary (Flow)

text

CopyEdit

User types in search box



localSearch update होता है



useEffect चलता है



allProducts से suggestions filter होते हैं



setSuggestions(filtered) → updates parent App.js state



suggestions list UI में show होती है (ul > li)